Java (langage)

Java est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au *SunWorld*.

La société Sun a été ensuite rachetée en 2009 par la société Oracle qui détient et maintient désormais Java.

La particularité et l'objectif central de Java est que les logiciels écrits dans ce langage doivent être très facilement portables sur plusieurs systèmes d'exploitation tels que Unix, Windows, Mac OS ou GNU/Linux, avec peu ou pas de modifications. Pour cela, divers plateformes et frameworks associés visent à guider, sinon garantir, cette portabilité des applications développées en Java.

Java

Date de première version	23 mai 1995
Paradigme	Orienté objet, structuré, impératif, fonctionnel
Auteur	Sun Microsystems
Développeurs	Oracle Corporation
Dernière version	10 (20 mars 2018)
Typage	Statique, fort, sûr, nominatif
Influencé par	Objective-C, C++, C#, Smalltalk, Eiffel[1], Ada 83, Mesa, Modula-3, Oberon, UCSD Pascal
A influencé	C#, J#, Ada 2005, Gambas, BeanShell, Clojure, ECMAScript, Groovy, JavaScript, PHP, Python[2], Scala, Seed7, Vala, Processing
Implémentations	Liste de JVM
Système d'exploitation	Multiplateformes
Licence	GNU GPL
Site web	www.java.com
Extensions de fichiers	.class, .jar, .jad, .java, .jmod

Sommaire

Aperçu

Le langage Java reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Néanmoins, Java a été épuré des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que les pointeurs et références, ou l'héritage multiple contourné par l'implémentation des interfaces. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs (nombres entiers, nombres à virgule flottante, etc.).

Java permet de développer des applications client-serveur. Côté client, les applets sont à l'origine de la notoriété du langage. C'est surtout côté serveur que Java s'est imposé dans le milieu de l'entreprise grâce aux servlets, le pendant serveur des applets, et plus récemment les JSP (JavaServer Pages) qui peuvent se substituer àPHP, ASP et ASP.NET.

Java a donné naissance à un système d'exploitation (JavaOS), à des environnements de développement (eclipse/JDK), des machines virtuelles (MSJVM **(en)**, JRE) applicatives multiplate-forme (JVM), une déclinaison pour les périphériques mobiles/embarqués (J2ME), une bibliothèque de conception d'interface graphique (AWT/Swing), des applications lourdes (Jude, Oracle SQL Worksheet, etc.), des technologies web (servlets, applets) et une déclinaison pour l'entreprise (J2EE). La portabilité du bytecode Java est assurée par la machine virtuelle Java, et éventuellement par des bibliothèques standard incluses dans un JRE. Cette machine virtuelle peut interpréter le bytecode ou le compiler à la volée en langage machine. La portabilité est dépendante de la qualité de portage des JVM sur chaque OS.

Historique

Origine du langage

Le langage Java est issu d'un projet de Sun Microsystems datant de 1990 : l'ingénieur Patrick Naughton n'était pas satisfait par le langage C++ utilisé chez Sun, ses interfaces de programmation en langage C, ainsi que les outils associés. Alors qu'il envisageait une migration vers NeXT, on lui proposa de travailler sur une nouvelle technologie et c'est ainsi que le **Projet Stealth** (furtif) vit le jour.

Duke, la mascotte de Java.

Le **Projet Stealth** fut rapidement rebaptisé **Green Project** avec l'arrivée de James Gosling et de Mike Sheridan. Ensemble, aidés d'autres ingénieurs, ils commencèrent à travailler dans un bureau de la rue Sand Hill à Menlo Park en Californie. Ils essayèrent d'élaborer une technologie pour le développement d'applications d'une nouvelle génération, offrant à Sun la perspective d'opportunités uniques.

L'équipe envisageait initialement d'utiliser le langage C++, mais l'abandonna pour différentes raisons. Tout d'abord, ils développaient sur un système embarqué avec des ressources limitées et estimaient que l'utilisation du C++ demandait un investissement trop important et que cette complexité était une source d'erreur pour les développeurs. L'absence de ramasse-miettes impliquait que la gestion de la mémoire devait être programmée manuellement, un défi mais aussi une source d'erreurs.

L'équipe était également insatisfaite des lacunes du langage C++ au niveau de la sécurité, de la programmation distribuée, du *multi-threading*. De plus, ils voulaient une plate-forme qui puisse être portée sur tout type d'appareils ou de plates-formes.

Bill Joy avait envisagé un nouveau langage combinant le meilleur du langage de programmation Mesa (en) et du langage C. Dans un article appelé *Plus loin (Further)*, il proposa à Sun que ses ingénieurs développent un environnement orienté objet basé sur le langage C++. À l'origine, Gosling envisageait de modifier et d'améliorer le langage C++, qu'il appelait C++ ++ --, mais l'idée fut bientôt abandonnée au profit du développement d'un nouveau langage de programmation qu'ils appelèrent ***Oak* (chêne)** en référence, selon la légende, à un arbre planté devant la fenêtre de leur bureau[réf. souhaitée].

L'équipe travailla avec acharnement[réf. souhaitée] et, à l'été 1992, ils furent capables de faire une démonstration constituée d'une plate-forme incluant le système d'exploitation Green[réf. souhaitée], le langage Oak (1992), les bibliothèques et le matériel. Leur première réalisation, présentée le 3 septembre 1992, fut la construction d'un PDA appelé **Star7** ayant une interface graphique et un agent intelligent appelé **Duke** pour prêter assistance à l'utilisateur

En novembre de la même année, le **Green Project** fut abandonné pour devenir **FirstPerson, Inc**, appartenant en totalité à Sun Microsystems et l'équipe fut relocalisée à Palo Alto. L'équipe **FirstPerson** était intéressée par la construction d'outils hautement interactifs et quand Time Warner publia un appel d'offres en faveur d'un décodeur multifonctions, FirstPerson changea d'objectif pour proposer une telle plate-forme.

Cependant, l'industrie de la télévision par câble trouva qu'elle offrait trop de possibilités à l'utilisateur et FirstPerson perdit le marché au profit de Silicon Graphics. Incapable d'intéresser l'industrie audiovisuelle, la société fut réintégrée au sein de Sun.

Java rencontre Internet

De juin à juillet 1994, après trois jours de remue-méninges avec John Gage, James Gosling, Joy, Naughton, Wayne Rosing et Eric Schmidt, l'équipe recentra la plate-forme sur le web. Ils pensaient qu'avec l'avènement du navigateur Mosaic, Internet était le lieu où allait se développer le même genre d'outil interactif que celui qu'ils avaient envisagé pour l'industrie du câble. Naughton développa comme prototype un petit navigateur web, WebRunner qui deviendra par la suite HotJava.

La même année le langage fut renommé Java après qu'on eut découvert que le nom **Oak** était déjà utilisé par un fabricant de carte vidéo.

Origine du nom Java

Le nom « Java » n'est pas un acronyme, il a été choisi lors d'un brainstorming[3] en remplacement du nom d'origine « Oak », à cause d'un conflit avec une marque existante, parce que le café (« java » en argot américain)[4] est la boisson favorite de nombreux programmeurs[5]. Le logo choisi par Sun est d'ailleurs une tasse de café fumant.

Lancement public de Java

En octobre 1994, HotJava et la plate-forme Java furent présentés pour Sun Executives. Java 1.0a fut disponible en téléchargement en 1994 mais la première version publique du navigateur HotJava arriva le 23 mai 1995 à la conférence **SunWorld**[6].

L'annonce fut effectuée par John Gage, le directeur scientifique de Sun Microsystems. Son annonce fut accompagnée de l'annonce surprise de Marc Andressen, vice-président de l'exécutif de Netscape que Netscape allait inclure le support de Java dans ses navigateurs. Le 9 janvier 1996, le groupe Javasoft fut constitué par Sun Microsystems pour développer cette technologie[7]. Deux semaines plus tard la première version de Java était disponible.

Avènement de Java 2

L'apparition de la version 1.2 du langage marque un tournant significatif : c'est en 2000 qu'apparait simultanément la déclinaison en deux plateformes Java :

- Java 2 Standard Edition (J2SE), plateforme avec les API et bibliothèques de bases, devenue depuis Java SE ;
- Java 2 Enterprise Edition (J2EE), extension avec des technologies pour le développement d'applications d'entreprise, devenue Java EE.

Sun les qualifie alors de plateforme Java 2 par opposition aux premières générations 1.0 et 1.1. Toutes les versions ultérieures, de J2EE 1.2 à Java SE ou Java EE 7 restent désignées sous le qualificatif de plateformes Java 2, bien que le '2' ait été depuis officiellement abandonné.[8]

Histoire récente

Utilisation Web

Côté client

Applets

Historiquement, la possibilité des navigateurs Web de lancer des applets Java était la seule solution pour afficher des applications clientes riches (RIA pour rich Internet application). Puis des technologies concurrentes ont émergé parmi lesquelles Macromedia Flash, le DHTML JavaScript, Silverlight basé sur XAML ou Xul.

Les applets sur le poste Client peuvent communiquer avec des servlets sur le Serveur, tout comme Javascript peut communiquer avec le Serveur au moyen d'AJAX. Flex utilise quant à lui la technologie Flash par le biais du Adobe Flash Player.

À une époque où JavaScript souffrait de problèmes de compatibilité inter-navigateurs, les applets Java avaient l'avantage de la portabilité car le portage d'interfaces complexes était difficile à assurer pour tous les navigateurs du marché.

Outre la retombée de la « mode » Java, les progrès faits dans les technologies concurrentes à Java ont amené la plupart des développeurs à se détourner des applets Java et des problèmes inhérents à cette technologie (incompatibilités entre les JVM, mauvaises performances, pauvreté des bibliothèques graphiques, complexité). Enfin, les navigateurs modernes n'incluent plus systématiquement l'environnement Java à cause de sa taille importante, et le taux de machines capables d'afficher des applets n'était plus que de 70 % en 2010, bien plus faible que pour Flash par exemple[8]. En 2010, la quasi-totalité des applications clients riches utilisent des technologies alternatives ; Flash pour l'essentiel mais aussi GWT.

Enfin, la perspective de l'arrivée prochaine de HTML5, destiné à embarquer de nombreuses fonctionnalités RIA et multimédia, rend également les applets caduques.

JavaFX

Avec l'apparition de Java 8 en mars 2014, JavaFX devient l'outil de création d'interface graphique ('GUI toolkit') officiel de Java, pour toutes les sortes d'application (applications mobiles, applications sur poste de travail, applications Web…), le développement de son prédécesseur Swing étant abandonné (sauf pour les corrections de bogues). JavaFX est une pure API Java (le langage de script spécifique qui lui a été un temps associé est maintenant abandonné). JavaFX contient des outils très divers, notamment pour les médias audio et vidéo, le graphisme 2D et 3D, la programmation Web, la programmation parallèle, etc.

Côté serveur

Avec les serveurs d'applications, on utilise des EJB pour encapsuler les classes définies précédemment. Ces éléments sont utilisés dans des architectures J2EE pour des applications multicouches. L'avantage qu'on tire de ce travail est de pouvoir cacher au client l'implémentation du code côté serveur.

Utilisation sur poste de travail

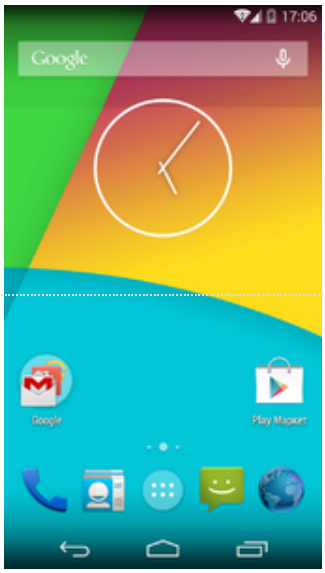
L'utilisation native du langage Java pour des applications sur un poste de travail restait jusqu'à présent relativement rare à cause de leur manque de rapidité. Cependant, avec l'accroissement rapide de la puissance des ordinateurs, les améliorations au cours des années 2000, de la machine virtuelle Java et de la qualité des compilateurs, plusieurs technologies ont gagné du terrain comme NetBeans et l'environnement Eclipse, les technologies de fichiers partagés LimeWire, Vuze (ex Azureus), et I2P. Java est aussi utilisé dans le programme de mathématiques MATLAB, au niveau de l'interface homme machine et pour le calcul formel. Les applications Swing apparaissent également comme une alternative à la technologie .NET.

Utilisation avec les mobiles

Oracle annonce début octobre 2012 à la conférence JavaOne sa volonté de proposer des solutions Java pour le domaine des logiciels embarqués, pour processeurs moins puissants que ceux habituellement disponibles sur les PC. Oracle fédère autour d'elle tout un éco-système d'entreprises spécialistes de ces segments de marchés, comme l'éditeur MicroEJ[9] ou encore STMicroelectronics qui propose du Java sur ses STM32[10] dont le cœur est un CortexM3/M4.

Java, notamment via Eclipse et NetBeans, offre déjà des environnements de développement intégrés pour mobile. Java est le principal langage utilisé pour développer des applications pour le système d'exploitation libre pour Mobile de Google : Android.

JavaFX peut aussi permettre l'utilisation de Java sur mobiles, bien que ce ne soit pas son objectif principal.



Android utilise beaucoup les technologies Java

OS Windows, Mac OS X et GNU/Linux

Microsoft a fourni en 2001 un environnement de travail de type Java, dénommé J++, avec ses systèmes d'exploitation avant la sortie de Windows XP. À la suite d'une décision de justice [réf. nécessaire], et au vu du non-respect des spécifications de ce langage, Microsoft a dû abandonner celui-ci et créer un nouveau langage, de nom C# (cf. chapitre « Indépendance vis-à-vis de la plate-forme » plus bas)

Beaucoup de fabricants d'ordinateurs continuent d'inclure un environnement JRE sur leurs systèmes Windows.

Java apparaît également comme un standard au niveau du Mac OS X d'Apple aussi bien que pour les distributions Linux. Ainsi, de nos jours, la plupart des utilisateurs peuvent lancer des applications Java sans aucun problème. Toutefois, sur ordinateur Apple, la distribution de Java 5 à Java 6 fut assurée directement par Apple, et non par Oracle[11]. Cette politique entraîna des retards et des restrictions de version :

> Ainsi J2SE 1.4 ne fut pas disponible avant Mac OS X v10.2 (nom de code *Jaguar*), J2SE 5.0 à partir de Mac OS X v10.4 (*Tiger*), Java SE 6 fonctionne uniquement sous Mac OS X v10.5 (*Leopard*) équipé de processeur Intel[12], et Java 7 exige un ordinateur Mac Intel exécutant la version Mac OS X v10.7.3 (*Lion*).

Passage sous licence open-source

Le 11 novembre 2006, le code source du compilateur `javac` et de la machine virtuelle HotSpot ont été publiés en Open Source sous la Licence publique générale GNU[13].

Le 13 novembre 2006, Sun Microsystems annonce le passage de Java, c'est-à-dire le JDK (JRE et outils de développement) et les environnements Java EE (déjà sous licence CDDL) et Java ME sous licence GPL d'ici mars 2007, sous le nom de projet OpenJDK[14].

En mai 2007, Sun publie effectivement OpenJDK sous licence libre. Cependant OpenJDK dépend encore de fragments de code non libre que Sun ne détient pas. C'est pourquoi la société Redhat lance en juin 2007 le projet IcedTea **(en)** qui vise à remplacer les fragments de code non libre et ainsi rendre OpenJDK utilisable sans aucun logiciel propriétaire. En juin 2008, le projet IcedTea a passé les tests rigoureux de compatibilité Java (TCK)[15]. IcedTea est donc une implémentation open-source des spécifications de Java. Sun, puis Oracle, garde toutefois le contrôle de la technologie par le biais d'un catalogue de brevets s'appliquant à Java, ainsi que par le maintien du TCK sous une licence propriétaire.

Acquisition par Oracle

La société Oracle a acquis en 2009 l'entreprise Sun Microsystems. On peut désormais voir apparaître le logo Oracle dans les documentations de l'api Java.

Le 12 avril 2010, James Gosling, le créateur du langage de programmation Java, démissionne d'Oracle pour des motifs qu'il ne souhaite pas divulguer. Il était devenu le directeur technologique de la division logicielle client pour Oracle.

Historique des versions

Le langage Java a connu plusieurs évolutions depuis le JDK 1.0 (*Java Development Kit*) avec l'ajout de nombreuses classes et packages à la bibliothèque standard. Depuis le J2SE1.4, l'évolution de Java est dirigée par le JCP (*Java Community Process*) qui utilise les JSR (*Java Specifications Requests*) pour proposer des ajouts et des changements sur la plate-forme Java. Le langage lui-même est spécifié par le JLS (*Java Language Specification*), les modifications du JLS étant gérées sous le code JSR 901[16].

Il faut noter que les évolutions successives du langage ne portent guère sur sa syntaxe -relativement stable depuis le début- mais principalement sur l'enrichissement de ses fonctions, avec l'embarquement et l'optimisation de bibliothèques logicielles (API) dans des domaines très variés de l'informatique : bases de données, gestion XML, informatique distribuée et web, multimédia, sécurité…

Numérotation des versions

Il faut distinguer la version du langage Java de celles des plateformes et du JRE :

- Le numéro majeur de version du langage (*par exemple* ***Java 5***) spécifiée par le JLS, est ainsi le numéro mineur pour le JRE ou la plateforme (par ex JRE 1.5 ou Java SE 1.5)[17] ;
- Le JRE et la plateforme Java SE sont eux identifiés depuis Java 5 par un même numéro de version : ainsi Java SE 1.6.0.43 et le JRE 1.6.0.43, sont tous deux conformes au langage Java 6, mais ont fait l'objet notamment de correctifs de sécurité[18] ;
- Java FX (orienté *RIA*), constitué essentiellement d'un sous-ensemble de JavaSE, suit également le même numéro de version : Java FX en est ainsi sa version 2.7 pour 1.7[19] (également compatible JRE 1.6) ;
- En revanche, les plateformes *Enterprise Edition* (Java EE) et *Mobile Edition* (Java ME) sont identifiées par le numéro de version de leur propre SDK ; ainsi début 2013 :
  - Java EE en était encore à la version 6 (Java EE 6 SDK Update 4), mais fonctionnait avec les JDK 6 ou 7[20] ; il n'est réellement passé à Java 7 que mi-juin 2013[21],
  - Java ME en est à sa version 3.3 (SDK v3.3)[22].

Versions successives

Deux versions peuvent parfois être proposées simultanément, telles que 8u65 & 8u66 : la différence consiste généralement en des corrections de bugs mineurs (sans incidence de sécurité notamment), pour lesquelles la mise à jour à la toute dernière version n'est pas critique et est de ce fait laissée au choix des administrateurs (JRE) ou développeurs (JDK).

Les versions publiques de Java peuvent être suivies de versions non publiques, dites Advanced, réservées à des usages commerciaux ; ainsi Java 1.6u45 est la dernière version publique de Java6, mais 6u113 l'ultime version disponible fin mars 2016.

Version	Last update	Dénomination JSE/JRE	Nom de code	Spécifications	JDK	Statut fin mars 2016	Période de maintenance	Support étendu[23]
1.0	1.0.2	Java 1.0	Oak	JSR 52	JDK 1.0.2	N'est plus soutenu de façon active	1996-2000	
1.1	8_16	Java 1.1		JSR 52	1.1.8_16	N'est plus soutenu de façon active	1997-2000	
1.2	2_017	J2SE 1.2	Playground	JSR 52	1.2.2_11	N'est plus soutenu de façon active	2000-2006	
1.3	1_29	J2SE 1.3	Kestrel	JSR 58	1.3.1_29	Obsolète	2000-2001	
1.4	2_30	J2SE 1.4	Merlin	JSR 59	1.4.2_30	Obsolète	2000-2008[24]	
1.5	0_22 à 0_85	J2SE 5.0	Tiger	JSR 176	1.5.0_22	Obsolète, 5u51 à 5u85 uniquement disponibles avec un support Oracle spécifique[25]	2002-2009[24]	Mai 2015
1.6	0_45 à 0_111	Java SE 6	Mustang	JSR 270	6u113	Obsolète, 6u51 à 6u111 uniquement disponibles avec un support Oracle spécifique[25]	2005-2013[24]	Décembre 2018
1.7	0_79 à 0_80	Java SE 7	Dolphin	JSR 336	1.7.0_79	Stable, actuel, version 1.7.0_79[26]	2011- 2015[24]	Juillet 2022
1.8	0_161	Java SE 8	Kenai[27]	JSR 337	1.8.0_161	Stable, actuel, version 1.8.0_161[28] proposée aux utilisateurs[29]	2014-sept2018	
9	9.0.1	Java SE 9			9.0.1		2018-?	
10		Java SE 10					2018-?	

Contenu et évolutions

Du JDK 1.0 au J2SE 1.4

- **JDK 1.0** (23 janvier 1996 - 211 classes et interfaces) — Version initiale[30].
- **JDK 1.1** (19 février 1997 - 477 classes et interfaces) — De nombreux ajouts[31] avec notamment :
  - une refonte complète du modèle événementiel AWT.
  - Les classes internes sont ajoutées au langage.
  - JavaBeans.
  - JDBC.
  - Java Remote Invocation (RMI).
- **J2SE 1.2** (9 décembre 1998 - 1 524 classes et interfaces) — Nom de code *Playground*. Cette version et les suivantes jusque J2SE 7.0 sont rebaptisées **Java 2** et la version nommée J2SE (*Java 2 Platform, Standard Edition*) remplace JDK pour distinguer la plate-forme de base de la version J2EE (*Java 2 Platform, Enterprise Edition*) et de la version J2ME (*Java 2 Platform, Micro Edition*). Plusieurs ajouts[32] dont :
  - le mot-clé `strictfp` (strict floating-point : virgule flottante stricte)

- la réflexion
- l'API graphique Swing est intégrée.
- Pour la première fois, la machine virtuelle Java de Sun inclut un compilateur « compilation à la volée » (*Just in Time*).
- Java Plug-in
- Java IDL, une implémentation de IDL pour l'interopérabilité avec CORBA.
- le *framework Collections*

- **J2SE 1.3** (8 mai 2000 - 1 840 classes et interfaces) — Nom de code *Kestrel*. Changements principaux[33] :
  - HotSpot JVM inclus (La machine virtuelle HotSpot sortit en avril 1999 pour la machine virtuelle du J2SE 1.2)
  - Changement pour les RMI pour être basé sur CORBA.
  - JavaSound
  - JNDI (*Java Naming and Directory Interface*) inclus de base (disponible auparavant comme extension)
  - JPDA (*Java Platform Debugger Architecture*)
- **J2SE 1.4** (6 février 2002 - 2 723 classes et interfaces) — Nom de code *Merlin*. Ce fut la première révision de la plate-forme sous JCP (*Java Community Process*)[34]. Les principaux changements[35] sont :
  - le mot-clé `assert` (Spécifié dans JSR 41[36].)
  - les expressions rationnelles modélisées en s'inspirant du langage Perl.
  - Le chaînage d'exception permet à une exception d'encapsuler l'exception de bas niveau d'origine. (Spécifié dans **(en)** JSR 51.)
  - API de journalisation (Spécifiée dans **(en)** JSR 47.)
  - l'API Image I/O pour lire et écrire des images dans des formats comme JPEG et PNG.
  - intégration d'un parser XML et du moteur XSLT nommé JAXP (Spécifié dans **(en)** JSR 5 et **(en)** JSR 63.)
  - intégration des extensions de sécurité JCE (Java Cryptography Extension), JSSE et JAAS.
  - Java Web Start (introduit pour la première fois en mars 2001 pour J2SE 1.3 - Spécifié dans **(en)** JSR 56.)

J2SE 5.0

Sorti le 30 septembre 2004 - 3 270 classes et interfaces), son nom de code est *Tiger*. Initialement numérotée 1.5, qui est toujours utilisé comme numéro de version interne[37]). Développé par **(en)** JSR 176, *Tiger* ajoute un nombre significatif de nouveautés[38] au langage :

- Intégration du composant logiciel Java Web Start dans l'environnement d'exécution Java (JRE)[39]
- Programmation générique — (Spécifié par **(en)** JSR 14)
- Metadata — également appelées annotations, permet au langage de construire des classes et des méthodes étiquetées avec des données additionnelles qui peuvent être utilisées en tant que méta-données (Spécifiée dans **(en)** JSR 175.)
- Autoboxing/unboxing — conversion automatique entre des types primitifs (comme le type `int`) et le Wrapper de classe correspondant (comme la classe Integer) (Spécifié dans **(en)** JSR 201).
- Énumérations — le mot-clé `enum` permet de créer une liste ordonnée de valeurs en gardant la sûreté du typage. Auparavant, ceci pouvait seulement être réalisé par des entiers constants (Spécifié dans JSR 201).
- Varargs — la syntaxe `Object…` utilisée dans une déclaration de méthode permet de spécifier un nombre variable d'arguments pour cette méthode. C'est un fonctionnement équivalent à la fonction « printf » en C.
- Imports statiques — Cette fonctionnalité permet d'utiliser les constantes d'une classe sans spécifier le nom de cette classe et sans passer par « l'*anti-pattern Constant Interface*» (c'est l'expression utilisée sur le site de Sun).
- Extension du `for` pour les boucles — la syntaxe du `for` est étendue avec une syntaxe spéciale pour itérer sur n'importe quel objet itérable comme un tableau, ou une collection en utilisant la syntaxe :

```
void displayWidgets (Iterable<Widget> widgets) {
    for (Widget w : widgets) {
        w.display();
    }
}
```

Cet exemple parcourt le contenu de l'objet widgets de la classe Iterable et contenant uniquement des références vers des objets de la classe Widget, assignant chacun de ces éléments à la variable w et ensuite appelle la méthode display() sur l'élément w (spécifié dans JSR 201). Une syntaxe similaire sera introduite en 2011 dans C++11.

En plus des changements au niveau du langage, des changements plus importants ont eu lieu au fil des années qui ont conduit des quelques centaines de classes dans le JDK 1.0 à plus de 3 000 dans J2SE 5.0. Des API entières, comme Swing ou Java2D, ont été ajoutées et beaucoup de méthodes de l'original JDK 1.0 ont été déclarées *deprecated* (c'est-à-dire déconseillées, elles pourraient être supprimées dans une version ultérieure de Java).

Java SE 6

Sorti le 11 décembre 2006, contient 3 777 classes et interfaces dans plus de 20 paquetages). Son nom de code *Mustang*[40]. Une version bêta est sortie le 15 février 2006, une autre bêta en juin 2006, une version « *release candidate* » en novembre 2006, et la version finale le 12 décembre 2006. Avec cette version, Sun remplace définitivement le nom J2SE par Java SE et supprime le .0 au numéro de version[41].

Cette version a été l'objet de nombreuses failles de sécurité et leurs mises à jour correctives, conduisant à la version 1.6.0_45 par Oracle et même 1.6.0_51 pour sa version Mac OS. C'est d'ailleurs là la dernière version de Java fonctionnant pour Mac OS X 10.6 et antérieurs.

Java SE 7

Sorti le 7 juillet 2011, contient 8 000[42] classes et interfaces). Son nom de code *Dolphin*. Il s'agit de la première version sous la licence GNU GPL.

Dès l'update 6 (7u6), l'édition standard Oracle de Java supportant de nouveau pleinement Mac OS X[43], les mises à jour pour cet OS ne sont plus prises en charge par Apple mais par Oracle. Toutefois cette version de Java n'est pas supportée par Mac OS X v10.6 : En effet certaines API requises par Java 7 ont bien été incluses par Apple dans Mac OS X 10.7.3, mais il n'est pas prévu qu'elles soient implémentées sur les précédentes versions de Mac OS[44]. La version 7u90 d'avril 2015 est la dernière mise à jour de Java 7 disponible publiquement[45].

Java 7 propose entre autres les nouveautés suivantes :

- la notation binaire ;
- le formatage numérique pour plus de lisibilité ;
- les switch avec des string ;
- l'inférence des types à la création d'instance pour éviter une redondance de syntaxe (de `List<String> lst = new ArrayList<>();`) ;
- le multicatch permettant de concaténer les exceptions catchées via des `|` ;
- java.nio (JSR 203) qui propose notamment une nouvelle interface Path, un système de parcours des répertoires, un service de watch…
- les tasks pour paralléliser les calculs jugés trop lourd ou trop coûteux ;
- l'autoboxing d'objets vers les types primitifs ;
- interface utilisateur : transparence des frames, bordures arrondies, gestion des évènements asynchrones via les secondary loops, les JLayers, les Painters, le nouveau style Nimbus…

Java SE 8

Nom de code Kenaï. Diverses releases en cours de développement du JDK sont disponibles au téléchargement dès l'automne 2013[46], et Java 8 sort mi-mars 2014 conformément à une roadmap présentée par Oracle dès mai 2013[47].

Une des nouveautés majeures de cette version est l'ajout des *lambdas*[48], entraînant une refonte d'une partie de l'API, notamment les collections et la notion de *stream*. Les autres ajouts notables incluent les optionnels, les implémentations par défaut au sein d'une interface, une refonte de l'API date, etc. En revanche la version Enterprise Edition (Java 8 EE) n'est pas attendue avant 2017.

La modularisation de la JVM avec le projet Jigsaw, initialement prévue pour cette version, est quant à elle reportée à la version 9[49], du fait notamment des failles de sécurité rencontrées par Java 6 dont Oracle a privilégié la correction en 2013 par rapport aux évolutions de Java.

Java SE 9

Initialement prévu pour 2015 mais reportée en partie à cause du projet Jigsaw, cette version est finalement sortie le 21 septembre 2017[50].

Java 9 intègre :

- le projet Jigsaw permettant de modulariser les modules chargés au sein du JDK ;
- le projet Kulla visant la création d'un shell pour Java sur le format read–eval–print loop (en) ;
- le projet Valhalla visant une amélioration des types Java ;
- un support natif du format JSON et de HTTP/2[51].

Java SE 10

Cette version est sortie le 20 mars 2018[52].

Cette nouvelle version intègre notamment :

- JEP 286[53] : inférence des types des variables locales;
- JEP 310[54]
- JEP 317[55]

Philosophie

Lors de la création du langage Java, il avait été décidé que ce langage devait répondre à cinq objectifs[56] :

1. simple, orienté objet et familier ;
2. robuste et sûr ;
3. indépendant de la machine employée pour l'exécution ;
4. très performant ;
5. compilé, multi-tâches et dynamique.

Langage orienté objet et familier

La première caractéristique, le caractère orienté objet (« OO ») et familier, fait référence à une méthode de programmation et de conception du langage et le fait qu'un programme écrit en Java ressemble assez fort à un programme écrit en C++.

Bien qu'il existe plusieurs interprétations de l'expression **orienté objet**, une idée phare dans ce type de développement est que les différents types de données doivent être directement associés avec les différentes opérations qu'on peut effectuer sur ces données. En conséquence, les données (appelées *Propriétés*) et le code les manipulant (appelé *Méthodes*) sont combinés dans une même entité appelée *Classe* d'objet. Le code devient logiquement découpé en petites entités cohérentes et devient ainsi plus simple à maintenir et plus facilement réutilisable, étant intrinsèquement modulaire.

D'autres mécanismes tels que l'*héritage* permettent d'exploiter toutes les caractéristiques d'une *Classe* précédemment écrite dans ses propres programmes sans même avoir à en connaître le fonctionnement interne — on n'en voit que l'*interface* (l'interface décrit les propriétés et les méthodes sans fournir le code associé). Java interdit la notion d'héritage depuis plusieurs classes parent sauf si elles sont des interfaces.

Dans la version 1.5 du langage ont été rajoutés les *génériques*, un mécanisme de polymorphisme semblable (mais différent) aux *templates* du langage C++ ou aux foncteurs d'OCaml. Les génériques permettent d'exprimer d'une façon plus simple et plus sûre les propriétés d'objets comme des conteneurs (listes, arbres…) : le type liste est alors considéré génériquement par rapport au type d'objet contenu dans la liste.

Mécanisme du ramasse-miettes

Cet élément contribue à la robustesse et à la performance des programmes, le ramasse-miettes (*Garbage Collector*) est appelé régulièrement et automatiquement pendant l'exécution du programme. Sur les systèmes multi-processeurs et/ou multi-cœurs, celui-ci emploie même des threads multiples à faible priorité afin de perturber le moins possible l'exécution du programme[57]. En outre, le programmeur peut au besoin suggérer de lancer le ramasse-miettes à l'aide de la méthode *System.gc()*.

Un grief récurrent à l'encontre de langages comme C++ est la lourde tâche d'avoir à programmer manuellement la gestion de la mémoire. En C++, la mémoire allouée par le programme pour créer un objet est désallouée lors de la destruction de celui-ci (par exemple par un appel explicite à l'opérateur *delete*). Si le programmeur oublie de coder la désallocation, ceci aboutit à une « fuite mémoire », et le programme en consomme de plus en plus. Pire encore, si par erreur un programme demande plusieurs fois une désallocation, ou emploie une zone de mémoire après avoir demandé sa désallocation, celui-ci deviendra très probablement instable et se plantera.

En Java, une grande partie de ces problèmes est évitée grâce au ramasse-miettes. L'espace mémoire nécessaire à chaque objet créé est alloué dans un tas de mémoire (en anglais : *memory heap*) réservé à cet usage. Le programme peut ensuite accéder à chaque objet grâce à sa référence dans le tas. Quand il n'existe plus aucune référence permettant d'atteindre un objet, le ramasse-miettes le détruit automatiquement — puisqu'il est devenu inaccessible — libérant la mémoire et prévenant ainsi toute fuite de mémoire.

Le ramasse-miettes emploie un algorithme de marquage puis libération (en anglais : *mark and sweep*[57]) qui permet de gérer les cas complexes d'objets se référençant mutuellement ou de boucles de références (cas d'une liste à chaînage double par exemple). En pratique, il subsiste des cas d'erreur de programmation où le ramasse-miettes considèrera qu'un objet est encore utile alors que le programme n'y accèdera plus, mais

dans l'ensemble, le ramasse-miettes rend plus simple et plus sûre la destruction d'objets en Java (en supprimant la nécessité de placer au bon endroit du code l'appel à l'opérateur *delete*).

Indépendance vis-à-vis de la plate-forme

L'indépendance vis-à-vis de la plate-forme signifie que les programmes écrits en Java fonctionnent de manière parfaitement similaire sur différentes architectures matérielles. La licence de Sun pour Java insiste ainsi sur le fait que toutes les implémentations doivent être compatibles. On peut ainsi théoriquement effectuer le développement sur une architecture donnée et faire tourner l'application finale sur toutes les autres.

Ce résultat est obtenu par :

- des bibliothèques standard fournies pour pouvoir accéder à certains éléments de la machine hôte (le graphisme, le multithreading, la programmation réseau…) exactement de la même manière sur toutes les architectures ;
- des compilateurs Java qui compilent le code source « à moitié » afin d'obtenir un **bytecode** (plus précisément le **bytecode Java**, un langage de type assembleur, proche de la machine virtuelle et spécifique à la plate-forme Java).

> Ce bytecode a ensuite vocation à être interprété sur une machine virtuelle Java (JVM en anglais), un programme écrit spécifiquement pour la machine cible qui interprète le bytecode Java et fait exécuter par la machine les instructions traduites en code natif.

Noter que même s'il y a explicitement une première phase de compilation, le bytecode Java est soit interprété, soit converti à la volée en code natif par un compilateur à la volée (*just in time*, JIT).

Types de compilations

Les premières implémentations du langage utilisaient une machine virtuelle interprétée pour obtenir la portabilité. Ces implémentations produisaient des programmes qui s'exécutaient plus lentement que ceux écrits en langage compilé (C, C++, etc.) si bien que le langage souffrit d'une réputation de faibles performances.

Des implémentations plus récentes de la machine virtuelle Java (JVM) produisent des programmes beaucoup plus rapides qu'auparavant, en utilisant différentes techniques :

- La première technique est de compiler directement en code natif comme un compilateur traditionnel, supprimant complètement la phase de bytecode. Des compilateurs Java tels que GNU Compiler for Java (GCJ) compilent ainsi directement le Java en code objet natif pour la machine cible. On obtient ainsi de bonnes performances, mais aux dépens de la portabilité : le code final produit par ces compilateurs ne peut de ce fait être exécuté que sur une seule architecture ;
- Une autre technique appelée compilation « juste-à-temps » ou « à la volée » (*just in time*, JIT), traduit le byte code en code natif durant la phase de lancement du programme ;
- Certaines machines virtuelles plus sophistiquées utilisent une *recompilation dynamique* durant laquelle la machine virtuelle analyse le comportement du programme et en recompile sélectivement certaines parties. La recompilation dynamique permet d'obtenir de meilleurs résultats que la compilation statique car les compilateurs dynamiques peuvent optimiser en fonction de leur connaissance de l'environnement cible et des classes qui sont utilisées.

> La compilation JIT et la recompilation dynamique permettent à Java de tirer profit de la rapidité du code natif sans perdre la portabilité.

Bilan de la portabilité Java

Après que Sun eut constaté que l'implémentation de Microsoft ne supportait pas les interfaces RMI et JNI, et comportait des éléments spécifiques à certaines plates-formes par rapport à sa plate-forme initiale, Sun déposa plainte en justice contre Microsoft[58], et obtint des dommages et intérêts (20 millions de dollars). Cet acte de justice renforça encore les termes de la licence de Sun. En réponse, Microsoft arrêta le support de Java sur ses plates-formes et, sur les versions récentes de Windows, Internet Explorer ne supporte pas les applets Java sans ajouter de plug-in. Cependant, Sun met à disposition gratuitement des environnements d'exécution de Java pour les différentes plates-formes Microsoft.

La portabilité est techniquement un objectif difficile à atteindre et le succès de Java en ce domaine est mitigé. Quoiqu'il soit effectivement possible d'écrire des programmes pour la plate-forme Java qui fonctionnent correctement sur beaucoup de machines cibles, le nombre important de plates-formes avec de petites erreurs et des incohérences a abouti à un détournement du slogan de Sun « *Write once, run anywhere* » (« Écrire une fois, exécuter n'importe où ») en « *Write once, debug everywhere* » (« Écrire une fois, déboguer partout ») !

L'indépendance de Java vis-à-vis de la plate-forme est cependant un succès avec les applications côté serveur comme les services web, les servlets et le Java Beans aussi bien que les systèmes embarqués sur OSGi, utilisant l'environnement **Embedded Java**.

Exécution sécurisée de code distant

La plate-forme Java fut l'un des premiers systèmes à offrir le support de l'exécution du code à partir de sources distantes. Une applet peut fonctionner dans le navigateur web d'un utilisateur, exécutant du code téléchargé d'un serveur HTTP. Le code d'une applet fonctionne dans un espace très restrictif, ce qui protège l'utilisateur des codes erronés ou mal intentionnés. Cet espace est délimité par un objet appelé *gestionnaire de sécurité*. Un tel objet existe aussi pour du code local, mais il est alors par défaut inactif.

Le gestionnaire de sécurité (la classe `SecurityManager`) permet de définir un certain nombre d'autorisations d'utilisation des ressources du système local (système de fichiers, réseau, propriétés système…). Une autorisation définit :

1. un code accesseur (typiquement, une applet — éventuellement signée — envoyée depuis un serveur web) ;
2. une ressource locale concernée (par exemple un répertoire) ;
3. un ensemble de droits (par exemple lire/écrire).

Les éditeurs d'applet peuvent demander un certificat pour leur permettre de signer numériquement une applet comme sûre, leur donnant ainsi potentiellement (moyennant l'autorisation adéquate) la permission de sortir de l'espace restrictif et d'accéder aux ressources du système local.

Éléments du langage

Voici un exemple d'un programme *Hello world* typique écrit en Java :

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Le fichier source porte presque toujours le nom de la classe avec l'extension ".java" (ici "HelloWorld.java", ce serait même obligatoire si la classe avait l'attribut **public** dans sa déclaration — la rendant alors accessible à tout autre programme). On peut compiler puis exécuter cet exemple sur le terminal avec les commandes suivantes (sous Linux) :

```
javac HelloWorld.java
export CLASSPATH=.
java HelloWorld
```

La ligne « `export CLASSPATH=.` » sert à indiquer à Java qu'il doit également chercher les programmes class dans le répertoire courant. Ce chemin peut également être spécifié au lancement du programme par l'option `-classpath` (ou `-cp` en abrégé) :

```
java -cp. HelloWorld
```

Mots réservés, primitifs et littéraux

```
abstract else instanceof static try boolean false
assert (1.4) enum (5.0) interface strictfp (1.2) volatile byte true
break extends native super while char
case final new switch double
catch finally package synchronized float
class for private this int
const (*) goto (*) protected throw long
continue if public throws short
default implements return transient void null
do import
```

Notes :

- (*) ces mots clefs sont réservés mais n'ont pas de signification pour le compilateur (il est juste interdit d'employer ce nom pour une variable par exemple) ;
- (1.2), (1.4) et (5.0) ces mots clefs ont été ajoutés avec la version indiquée du langage.

(source)

Classe, attributs, méthodes

Notion de classe

Une classe est la description de données appelées attributs, et d'opérations appelées méthodes. Il s'agit d'un modèle de définition pour des objets ayant le même ensemble d'attributs, et le même ensemble d'opérations. À partir d'une classe on peut créer un ou plusieurs objets par instanciation ; chaque objet est une instance d'une seule classe.

Visibilité :

- Public : le mot `class` est alors précédé de `public`, tout utilisateur qui importe le paquetage peut utiliser la classe. Dans ce cas elle doit être définie dans un fichier qui a pour nom le nom de la classe.
- Privé : le mot `class` est alors précédé de `private`, seules des classes définies dans le même fichier peuvent utiliser cette classe.
- Paquetage : le mot `class` n'est pas précédé de mot particulier, toutes les classes du paquetage peuvent utiliser la classe.

Notion d'attribut

Un attribut se définit en donnant son type, puis son nom, et éventuellement une partie initialisation.

Visibilité :

- Public: sa définition est précédée de `public`, et il peut être utilisé par tout utilisateur de la classe.
- Privé: sa définition est précédée de `private`, et il ne peut être utilisé qu'à l'intérieur de la classe
- Protégé: sa définition est précédée de `protected`, et il ne peut être utilisé qu'à l'intérieur de la classe, ou des classes dérivées.
- Paquetage: aucun mot particulier ne précède sa définition, ainsi il peut être utilisé dans toute classe du même paquetage.

Notion de méthode

Une méthode est définie par :

- Son type de retour : type de la valeur retournée par la méthode. Si la méthode ne retourne pas de valeur le type spécifié est alors `void`.
- Son nom
- Ses paramètres : les paramètres sont spécifiés par leur type et leur nom et sont séparés par des virgules.

Visibilité :

- Public: sa définition est précédée de `public`, et elle peut être utilisée par tout utilisateur de la classe.
- Privé: sa définition est précédée de `private`, et elle ne peut être utilisée qu'à l'intérieur de la classe
- Protégé: sa définition est précédée de `protected`, et elle ne peut être utilisée qu'à l'intérieur de la classe, ou des classes dérivées.
- Paquetage: aucun mot particulier ne précède sa définition, ainsi la méthode peut être utilisé dans toute classe du même paquetage.

Types

Nom	Taille en octets lors des calculs	Valeur par défaut	Valeurs possibles
boolean	Un seul bit suffit, mais on réserve souvent un octet pour les stocker.	`false`	`true`, `false`
byte	1	`0`	entiers compris entre -128 et +127 (-2^7 et 2^7-1)
short	2	`0`	entiers compris entre −32 768 et 32 767 (-2^{15} et 2^{15}-1)
int	4	`0`	entiers compris entre −2 147 483 648 et +2 147 483 647 (-2^{31} et 2^{31}-1)
long	8	`0`	entiers compris entre −9 223 372 036 854 775 808 et 9 223 372 036 854 775 807 (-2^{63} et 2^{63}-1)
char	2	`'\u0000'`	Ensemble des valeurs Unicode (valeurs de U+0000 à U+FFFF, 4 chiffres obligatoires après `'\u'`) Les 128 premiers caractères sont les codes ASCII et se notent entre apostrophes : `'a'`, `'1'`, `'\`, `'\n'`.
float	4	`0.0`	Ensemble des nombres [−3,402 823 47 ×10^{38} .. −1,402 398 46 ×10^{-45}], 0, [1,402 398 46 ×10^{-45} .. 3,402 823 47 ×10^{38}]
double	8	`0.0`	Ensemble des nombres [−1,797 693 134 862 315 70 ×10^{308} .. −4,940 656 458 412 465 44 ×10^{-324}], 0, [4,940 656 458 412 465 44 ×10^{-324} .. 1,797 693 134 862 315 70 ×10^{308}]
Object	Dépendant de la machine virtuelle	`null`	

Le tableau ci-dessus recense les types de base, cependant il existe en Java d'autres types qui sont des objets et sont à utiliser en tant que tel. Par exemple pour définir un entier on peut utiliser le type 'Integer' dont la valeur d'initialisation par défaut vaudra `null`.

Pour instancier une variable, la syntaxe (ici la même qu'en C) est la suivante :

```
NomDuType maVariable;
```

maVariable est alors allouée sur la pile.

Les collections d'objets

Il est souvent nécessaire de stocker de nombreuses données dans des collections : liste d'achats, notes des élèves, etc. Les collections peuvent être consultées, modifiées, on peut les trier, les recopier, les supprimer, etc. Elles peuvent avoir une taille fixe ou variable.

Les collections à taille fixe sont moins lourdes que les collections à taille variable.

Collections de taille fixe

- Les tableaux :

  Comme en C, les indices d'un tableau de taille **n** vont de 0 à n-1. Un des plus grands avantages des tableaux est l'accès en temps constant O(1), à chaque case du tableau.
  Les tableaux sont des objets en Java. Pour initialiser un tableau `tab` d'entiers de 10 cases, on peut écrire (entre autres) :

```
int[] tab = new int[10];
```

ou :

```
int tab[] = new int[10];
```

ou :

```
int tab[] = {0, 0, 0, 0, 0,
    0, 0, 0, 0, 0};
```

Les indices de `tab` vont de 0 à 9, et il ne pourra pas être redimensionné.
Chaque élément de `tab` est initialisé à sa valeur par défaut, ici 0 car il s'agit d'un tableau d'entiers (`int`). La valeur par défaut d'un objet est `null`, celle d'un booléen est `false`.

Pour lire ou modifier l'élément de `tab` d'indice **i**, pour 0 ≤ i ≤ 9, on utilise *tab[i]*.

Exemple : Pour attribuer la valeur 2 à une case de `tab`, on écrit : `tab[i] = 2;`

La propriété **length** (exemple : `tab.length`) fournit le nombre d'éléments du tableau. Si on essaie de modifier un élément qui n'existe pas, par exemple avec `tab [10] = 4;`, une exception *IndexOutOfBoundsException* est déclenchée (notre tableau `tab` ne contient que 10 cases, numérotées de 0 à 9 inclus).

Collections de taille variable

La classe abstraite`AbstractList`est fournie pour implémenter les collections à taille variable.

Pour initialiser une ArrayList il faut importer la classe `java.util.ArrayList`et écrire `liste = new ArrayList<T>();` ou `liste = new ArrayList<>();`depuis le JDK 7.0.

Depuis le JDK 1.5, on a la possibilité d'indiquer le type des éléments contenus dans une ArrayList : Entiers, chaînes de caractères ou autres objets.

- Cette collection est vide après l'appel du constructeur ;
- On peut ajouter autant d'éléments que l'on veut.

Pour ajouter un élément on écrit`liste.add(Objet);`

Pour accéder à un élément de l'ArrayList :`liste.get(index);`

Pour connaître le nombre d'éléments que contient une liste `liste.size();`

Pour supprimer un élément :`liste.remove(index);`les éléments qui suivent l'élément supprimé seront décalés à gauche.

Structures de contrôle

Boucles

Bien qu'elles aient toutes un rôle similaire, chaque boucle est pourtant adaptée à une situation :

- Structure **tant que** (adaptée pour e**f**fectuer des opérations tantqu'une condition est remplie) :

```
while (<expression booléenne>) {
   instruction(s)
}
```

- Structure **faire … tant que** (comme la structure **tant que** mais la première itération est exécutée quelle que soit la condition, pour les autres itérations la condition doit être remplie) :

```
do {
   instruction(s)
}
while (<expression booléenne>);
```

- Structure **pour** (adaptée lorsqu'une collection doit être parcourue en totalité pour traitement) :

```
for (<initialisation> ; <condition de poursuite> ; <expression d'incrémentation>) {
   instruction(s)
}
```

- Structure **pour chaque** (simplification du for en for each, dans laquelle l'expression doit être un tableau ou une collection) :

```
for (type variable : <expression>) {
   instruction(s)
}
```

- Structure **pour** (Java 1.5) :

```
for (<Objet récupérant l'occurrence suivante de la collection> : <collection d'objets>) {
   instruction(s)
}
```

Structures conditionnelles

- Structure **si** : condition simple

```
if (<expression booléenne>) {
   instruction(s)
}
```

- Structure **si … sinon** : condition avec alternative unique

```
if (<expression booléenne>) {
    instruction(s)
}
else {
    instruction(s)
}
```

- Structure **si … ou si … ou si …** : condition avec alternatives multiples

```
if (<expression booléenne>) {
    instruction(s)
}
else if (<expression booléenne>) {
    instruction(s)
}
else if (<expression booléenne>) {
    instruction(s)
}
else {
    instruction(s)
}
```

- Structure **atteindre … cas x … cas y …** : embranchement vers un bloc d'instructions énuméré.

```
switch (<expression>) {
   case <constante>:
        instruction(s)
        break;
   case <constante>:
       instruction(s)
       break;
   [...]
   default:
        instruction(s)
        break;
}
```

Le choix d'exécution des instructions est déterminé par l'expression suivant `switch` dont le type peut être entier (`int`, `char`, `byte` ou `short` ou classes enveloppes correspondantes), énuméré (`enum`) ou `String` (chaîne de caractères, depuis Java 7 seulement).

Remarque : L' *opérateur conditionnel* ou *opérateur ternaire* `? :` peut permettre d'éviter l'utilisation d'une instruction conditionnelle. Une expression conditionnelle aura l'une ou l'autre valeur après test de la condition booléenne :

```
<expression booléenne> ? <valeur si vrai> : <valeur si faux>
```

Branchements inconditionnels

- L'instruction **`break`** fait sortir immédiatement de la boucle en cours (`for`, `while`, `do`), mais permet également de sortir d'une clause contenue dans un `switch`. Si le `break` est omis, l'exécution du `switch` se poursuit de `case` en `case`.
- L'instruction **`continue`** termine l'itération en cours et continue à la prochaine. Son usage ainsi que celui de `break` dans les boucles est controversé (de la même manière que `goto` dans d'autres langages structurés). Elle tendrait à favoriser un type de programmation non structurée (*programmation spaghetti*).
- L'instruction **`return`** termine une méthode. Avec `return uneValeur`, une valeur sera renvoyée à la méthode appelante.

Traitement des exceptions

```
try {
      instruction(s)
}
catch (<type exception> <variable>) {
      instruction(s)
}
[...]
finally {
      instruction(s)
}
```

Le bloc de code **finally** sera exécuté quel que soit le résultat lorsque le programme sortira du bloc **try-catch**.

Voici un exemple de capture d'une exception :

```
FileOutputStream  fos = null;

try {
     //Chacune de ces deux instructions peut générer une exception
     // création d'un flux pour écrire dans un fichier
     fos = new FileOutputStream(...);
     // écriture de données dans ce flux
     fos.write(...);
}
catch (IOException e) {
     //Gestion de l'erreur de création ou d'écriture dans le flux
     e.printStackTrace();
}
finally{
    //Cette section de code est toujours exécutée, qu'il y ait une exception ou pas
    // fermeture du flux s'il a été ouvert
    if(fos != null) fos.close();
}
```

Cet exemple permet d'illustrer le mécanisme des exceptions en Java. Dans le cas d'une erreur d'entrée/sortie dans le bloc **try**, l'exécution reprend dans le bloc **catch** correspondant à cette situation (exception de type **IOException**).

Dans ce bloc **catch**, la variable **e** référence l'exception qui s'est produite. Ici, nous invoquons la méthode **printStackTrace()** qui affiche dans la console des informations sur l'exception qui s'est produite : nom, motif, état de la pile d'appels au moment de la levée de l'exception et, éventuellement, numéro de ligne auquel l'erreur s'est produite.

Le bloc **finally** est ensuite exécuté (ici pour refermer les ressources utilisées). Il ne s'agit ici que d'un exemple, l'action à mettre en œuvre lorsqu'une exception survient dépend du fonctionnement général de l'application et de la nature de l'exception.

Types génériques

Un type générique est autrement appelé un Template, il prend un ou plusieurs autres types en arguments. Le type passé en paramètre est déterminé lors de l'instanciation.

Cela permet notamment dans le cadre des ArrayList d'éviter les transtypages.

```
public class ArrayList<E> {
   ArrayList<String> al = new ArrayList<String>();
}
```

Ces types génériques ne sont utilisés qu'à la compilation, et non directement dans le bytecode.

Différence avec le C++ : les templates en C++ dupliquent une classe pour chaque type. Java, au contraire, agit au moment de la compilation comme si on avait dupliqué les classes de ces types intrinsèques mais ne traite en réalité qu'avec une seule classe.

Codage du code source

Les spécifications du langage Java précisent qu'il est formé de caractères au format UTF-16, ce qui permet l'utilisation dans le code source de tous les caractères existant dans le monde :

```
public class HelloWorld {
   private String text = "hello world";
}
```

Pour assurer la portabilité entre plates-formes, les noms de classes devraient néanmoins être formés uniquement de caractères ASCII.

Opérateur de comparaison

Opérateur	Signification
==	Egal
!=	Différent
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

Ne pas confondre l'opérateur de test `==` et l'opérateur d'affectation `=`. Pour combiner des conditions il est possible d'utiliser des opérateurs supplémentaires : le « ET » logique `&&` et le « OU » logique `||` ; il existe également le « NON » logique `!` qui permet d'inverser le résultat d'une condition.

Environnements de développement

JavaStyle

Les JavaStyle sont des conventions de programmation en langage Java définies par Sun. Le respect de conventions strictes assure une homogénéité dans le code source d'une application développée par toute une équipe et favorise la diffusion du code source auprès d'une communauté partageant les mêmes conventions de codage.

Le *lower camel case* est utilisé pour les noms de méthodes et de variables.

Frameworks et API

Sun fournit un grand nombre de *frameworks* et d'API afin de permettre l'utilisation de Java pour des usages très diversifiés.

On distingue essentiellement quatre grands *frameworks* :

- Java SE (anciennement J2SE) : Ce *framework* est destiné aux applications pour poste de travail ;
- Java EE (anciennement J2EE) : Ce *framework* est spécialisé dans les applications serveurs. Il contient pour ce faire un grand nombre d'API et d'extensions ;
- Java ME (anciennement J2ME) : Ce *framework* est spécialisé dans les applications mobiles ;
- Java FX (à ne pas confondre avec JavaFX) : Ce *framework* est spécialisé dans les applications liées aux cartes à puces et autres SmartCards. Il recouvre notamment l'ancien Java Card.

La persistance est fondée sur les standards :

- JDBC (*Java DataBase Connectivity*) ;
- JDO (*Java Data Objects*) ;
- EJB (*Enterprise Java Beans*).

On trouve toutefois de nombreuses autres technologies, API et extensions optionnelles pour Java :

- Java Media Framework (en) : framework multimédia, contenant notamment les API Java2D, Java 3D, JavaSound, Java advanced Imaging ;
- Java Telephony API (en) ;
- Java TV (en) ;
- JXTA : Système de peer-to-peer reposant sur Java ;
- Jini ;
- JAIN ;
- Java Dynamic Management Kit (en) (JMDK) ;
- JavaSpeech ;
- JMI ;
- JavaSpaces.

Outils de développement

La programmation peut se faire depuis une invite de commande en lançant un compilateur Java (souvent nommé `javac`), mais pour avoir plus de confort, il est préférable d'utiliser un environnement de développement intégré ou IDE, certains sont gratuits. Par exemple :

- BlueJ ;
- CodeWarrior ;
- Eclipse ;
- IntelliJ IDEA ;
- JBuilder ;
- JCreator ;
- jDeveloper 2 ;
- NetBeans ;
- Xcode.

Automatisation

Un programme Java peut être produit avec des outils qui automatisent le processus de construction (c'est-à-dire l'automatisation de certaines tâches faisant appel à un nombre potentiellement grand de dépendances comme l'utilisation de bibliothèques, la compilation, la génération d'archives, de documentation, le déploiement, etc.). Les plus utilisés sont :

- Apache Ant (génération portable, décrite en XML) ;
- Apache Maven (génération portable, décrite en XML) ;
- Gradle (génération portable, en utilisant le langage Groovy) ;
- SCons (génération portable, en utilisant le langage Python). Exemple :

```
Java(target = 'classes', source = 'src')
      Jar(target = 'test.jar', source = 'classes')
```

Résultats :

```
    % scons -Q
    javac -d classes -sourcepath src src/Exemple1.java src/Exemple2.java src/Exemple3.java
    jar cf test.jar classes
```

Notes et références

1. *In Java 5.0, several features (the enhanced for loop, autoboxing, varargs, annotations and enums) were introduced, after proving themselves useful in the similar (and competing) language [1] (http://www.barrycornelius.com/papers/java5/)[2] (http://www.levenez.com/lang/)[3] (http://eclipsezone.com/eclipse/forums/t54318.html)*
2. http://www.python.org/dev/peps/pep-0318/
3. « So why did they decide to call it Java? » (http://www.javaworld.com/article/2077265/core-java/so-why-did-they-decide-to-call-it-java-.html), sur *JavaWorld*, 4 octobre 1996
4. *Core Web Programming* (https://books.google.fr/books?id=q45_UDI77PoC&pg=PR162&dq=java+language+coffee&hl=fr&sa=X&ved=0ahUKEwiK3Pq6qLfPAhWGSRoKHWnGDUoQ6AEIRzAE#v=onepage&q=java%20language%20coffee&f=false), Marty Hall, Larry Brown
5. *Object-oriented Programming with Java* (https://books.google.fr/books?id=271wpK2CQ0EC&pg=PR2&dq=java+language+coffee&hl=fr&sa=X&ved=0ahUKEwiK3Pq6qLfPAhWGSRoKHWnGDUoQ6AEIKjAA#v=onepage&q=coffee&f=false), Barry J. Holmes, Daniel T. Joyce
6. (en) *Java Technology: The Early Years* (https://web.archive.org/web/20100105045840/http://java.sun.com/features/1998/05/birthday.html).

7. **(en)** *Sun Microsystems Announces Formation Of Javasoft* (http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960109.14048.html).
8. http://riastats.com/
9. MicroEJ (http://www.microej.com/)
10. Java sur STM32 (http://www.stm32java.com)
11. Apple Java 6 pour Mac OS X (http://www.java.com/fr/download/faq/java_6.xml) FAQ http://www.java.com/fr/download
12. Technical Notes TN2110 (https://developer.apple.com/library/mac/#technotes/tn2002/tn2110.html#TABLES) developer.apple.com
13. **(en)** *"Q: What components of the JDK software are you open sourcing today? A: We're open sourcing the Java programming language compiler ("javac"), and the Java HotSpot virtual machine." Free and Open Source Java FAQ* (http://www.sun.com/software/opensource/java/faq.jsp#b2) the source is being released via the OpenJDK (https://openjdk.dev.java.net/) project.
14. « Sun Open Sources Java Platform » (http://www.sun.com/smi/Press/sunflash/2006-11/sunflash.20061113.1.xml) Sun Microsystems, 13 novembre 2006 (consulté le 13 novembre 2006)
15. http://blog.softwhere.org/archives/196
16. **(en)** *The Java Community Process(SM) Program - JSRs : Java Specification Requests - detail JSR# 901* (http://www.jcp.org/en/jsr/detail?id=901)
17. Java Naming (http://www.java.com/en/about/brand/naming.jsp) http://www.java.com.
18. Versions de Java et correctifs de sécurité (https://www.java.com/fr/download/faq/release_dates.xml).
19. **(en)** « JavaFX GA downloads » (http://www.oracle.com/technetwork/java/javafx/downloads/index.html)
20. http://www.oracle.com/technetwork/java/javaee/downloads/index.html
21. Oracle Announces Availability of Java Platform Enterprise Edition 7 (http://www.oracle.com/us/corporate/press/1957557) Oracle Press Release June 12, 2013.
22. **(en)** « Java ME SDK Downloads » (http://www.oracle.com/technetwork/java/javame/javamobile/download/sdk/index.html)
23. Oracle Java SE Commercial Offering Releases (http://www.oracle.com/technetwork/java/javase/eol-135779.html#java-commercial-offerings) oracle.com java commercial offerings
24. Oracle Java SE Support Roadmap, 13 mai 2013 http://www.oracle.com/technetwork/java/eol-135779.html
25. Oracle releases Java 7u25, no 6u51 (https://www.404techsupport.com/2013/06/java-7u25)
26. http://www.oracle.com/technetwork/java/javase/7u79-relnotes-2494161.html
27. https://jdk8.java.net « Copie archivée » (https://web.archive.org/web/20140829134921/https://jdk8.java.net/) (*version du 29 août 2014 sur l'Internet Archive*)
28. http://www.oracle.com/technetwork/java/javase/8u77-relnotes-2944725.html
29. 18/03/2014 (https://blogs.oracle.com/java/java-se-8-is-now-available)
30. **(en)** communiqué de presse (http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.html)
31. **(en)** communiqué de presse (http://www.sun.com/smi/Press/sunflash/1997-02/sunflash.970219.0001.html)
32. **(en)** communiqué de presse (http://www.sun.com/smi/Press/sunflash/1998-12/sunflash.981208.9.html)
33. **(en)** communiqué de presse (http://www.sun.com/smi/Press/sunflash/2000-05/sunflash.20000508.3.html) - **(en)** liste complète des changements (http://java.sun.com/j2se/1.3/docs/relnotes/features.html)
34. **(en)** JSR 59 (http://www.jcp.org/en/jsr/detail?id=59)
35. **(en)** communiqué de presse (http://www.sun.com/smi/Press/sunflash/2002-02/sunflash.20020206.5.html) - **(en)** liste complète des changements (http://java.sun.com/j2se/1.4.2/docs/relnotes/features.html)
36. **(en)** JSR 41 (http://www.jcp.org/en/jsr/detail?id=41)
37. **(en)** Version 1.5.0 or 5.0? (http://java.sun.com/j2se/1.5.0/docs/relnotes/version-5.0.html)
38. **(en)** communiqué de presse (http://www.sun.com/smi/Press/sunflash/2004-09/sunflash.20040930.1.html)
39. Qu'est-ce que le logiciel Java Web Start (http://www.java.com/fr/download/faq/java_webstart.xml) http://www.java.com
40. **(en)** JSR 270 (http://www.jcp.org/en/jsr/detail?id=270)
41. **(en)** Java Naming (http://www.java.com/en/about/brand/naming.jsp)
42. http://docs.oracle.com/javase/7/docs/api
43. Oracle met à jour Java SE et JavaFX pour OS X, ARM et Linux (http://www.lemondeinformatique.fr/actualites/lire-oracle-met-a-jour-java-se-et-javafx-pour-os-x-arm-et-linux-50072.html) Le Monde Informatique
44. Informations et configuration minimale requise pour l'installation et l'utilisation de Java 7 pour Mac (http://www.java.com/fr/download/faq/java_mac.xml#java%20available) http://www.java.com
45. Auto-update Notice and End of Public Updates for Oracle JDK 7 (http://www.oracle.com/technetwork/java/eol-135779.html) oracle.com, May 14, 2015
46. JDK™ 8 Early Access Releases (https://jdk8.java.net/download.html) java.net.
47. JDK8 Schedule and status (http://openjdk.java.net/projects/jdk8) OpenJDK.
48. Everything about Java 8 (http://www.infoq.com/fr/news/2013/09/everything-about-java-8)
49. [4] (http://www.lemondeinformatique.fr/actualites/lire-oracle-reporte-le-projet-jigsaw-systeme-de-modules-pour-java-49797.html)
50. Mickael Baron, « Java 9 est disponible, la plateforme se met aux modules : tour d'horizon des nouveautés », *Developpez.com*, 21 septembre 2017 (lire en ligne (https://java.developpez.com/actu/160955/Java-9-est-disponible-la-plateforme-se-met-aux-modules-tour-d-horizon-des-nouveautes/))

51. Java 9 : ce que l'on sait (http://www.journaldunet.com/developpeur/java-j2ee/java-9-0814.shtml) Journal du net - 16 juin 2015.
52. « JDK 10 » (http://openjdk.java.net/projects/jdk/10/), sur *openjdk.java.net* (consulté le 1er avril 2018)
53. « JEP 286: Local-Variable Type Inference » (http://openjdk.java.net/jeps/286), sur *openjdk.java.net* (consulté le 1er avril 2018)
54. « JEP 310: Application Class-Data Sharing » (http://openjdk.java.net/jeps/310), sur *openjdk.java.net* (consulté le 1er avril 2018)
55. « JEP 317: Experimental Java-Based JIT Compiler » (http://openjdk.java.net/jeps/317), sur *openjdk.java.net* (consulté le 1er avril 2018)
56. **(en)** Design Goals of the Java Programming Language (http://java.sun.com/docs/white/langenv/Intro.doc2.html)
57. Voir par exemple les détails du fonctionnement et la description des options *-XX:+UseParallelGC* et *-XX:+UseConcMarkSweepGC* de la JRE de Sun **(en)** Java SE Virtual Machine Garbage Collection Tuning (http://java.sun.com/javase/technologies/hotspot/gc/gc_tuning_6.html)
58. Microsoft poussé à parler Java (http://www.liberation.fr/economie/010143513-microsoft-pousse-a-parler-java) Libération - 26/12/2002

Voir aussi

Bibliographie

- Ken Arnold, James Gosling et David Holmes, *Le Langage Java*, 2001 (ISBN 978-2-7117-8671-8)
- **(en)** Bruce Eckel, *Thinking in Java*, Prentice-Hall, 2002 (ISBN 978-0-13-100287-6). version téléchargeable version électronique en ligne traduction francophone : penser en java
- Alexandre Brillant, *Java 5*, 2006 (ISBN 978-2-7460-3170-8)
- Jean Brondeau, *Introduction à la programmation objet en Java*, Dunod, Paris, 1999 (ISBN 978-2-10-004106-0)
- Annick Fron, *Architectures réparties en Java*, 2007 (ISBN 978-2-10-051141-9)
- Le tutoriel en ligne How To Program In JAVA [lire en ligne]

Sur les autres projets Wikimedia :

Java, sur Wikimedia Commons

Java, sur Wikiversity *Java*, sur Wikibooks

Articles connexes

- Plate-forme Java
- Machine virtuelle Java
- Bytecode Java
- Java et logiciel libre
- Servlet
- Navigateur HotJava
- GNU Compiler Collection (inclut un compilateur de code Java vers code natif, GCJ)
- .properties, manière de stocker les variables de configuration
- Processing
- Java version history **(en)**

Liens externes

- Site officiel
- **(en)** http://docs.oracle.com/javase/1.5.0/docs : Schéma de l'architecture logicielle de Java
- http://java.developpez.com/ : Communauté de développeurs Java
- http://how-to-program-in-java.com/ : tutoriel Java
- http://news.humancoders.com/t/java : Actualité pour développeurs Java
- **(en)** http://www.java.net : Site collaboratif officiel avec un wiki